|  |
| --- |
| Circle Language Spec: System Objects |

## System Command Call Notations

### Introduction

There might be several notations for calling a system command.

Direct calls to **Get**, **Set** and **Use** commands might not usually be seen. Those system commands may usually be called indirectly by *assignment* commands.

But being introduced to using **Get** and **Set** commands, instead of a single assignment call, might give a lower-level view on assignment. This lower-level perspective might be of use when wanting to use alternative forms of assignment, that do not seem to fit into an assignment notation.

### Notational Choices

#### Distinguishing Object-Bound and Reference-Bound Aspects

There are system commands that apply to object references and to the object itself. System commands, that apply to an object reference are displayed here with a parent around the targeted reference:

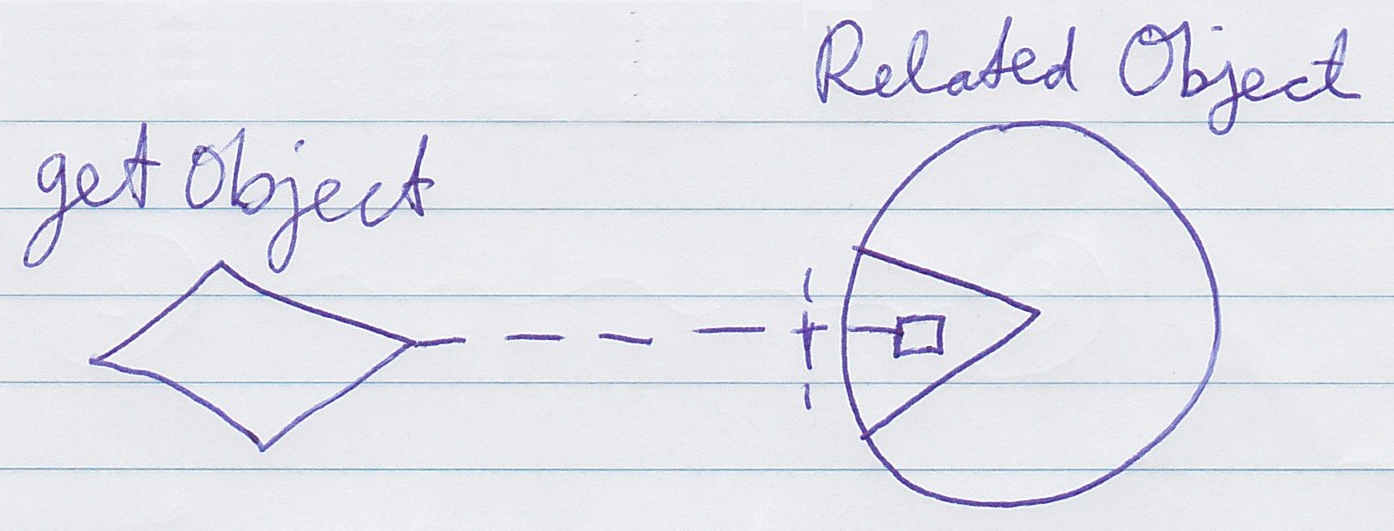


System commands, that apply to the object itself are displayed without a parent around the targeted object.



#### System Interface Call Notation

One of the notations for system command calls is calling upon the command definition in the system interface:



This might be a regular way to display a command call in Circle, but now a call to a command inside the system interface.

There could be multiple ways to display the content of the system interface. They might depend on taste, application or needs. This article may go for a system interface that looks relatively simple with often a loose square in it representing a system command.

### System Interface Calls versus Assignment

System command calls to **Get**, **Set** or **Use** might not often be visible, because they might usually be implied by assignment calls. But for alternative types of assignment, that do not have a standard assignment notation, you can explicitly display **Get**, **Set** and **Use** calls. When a simpler notation does not suffice, displaying **Get**, **Use** or **Set** command explicitly might be needed. This basically only happens in unusual cross-aspect assignments. In abstract diagram expression though, you can always use the explicit notations to simply make a point.

### System Interface Calls

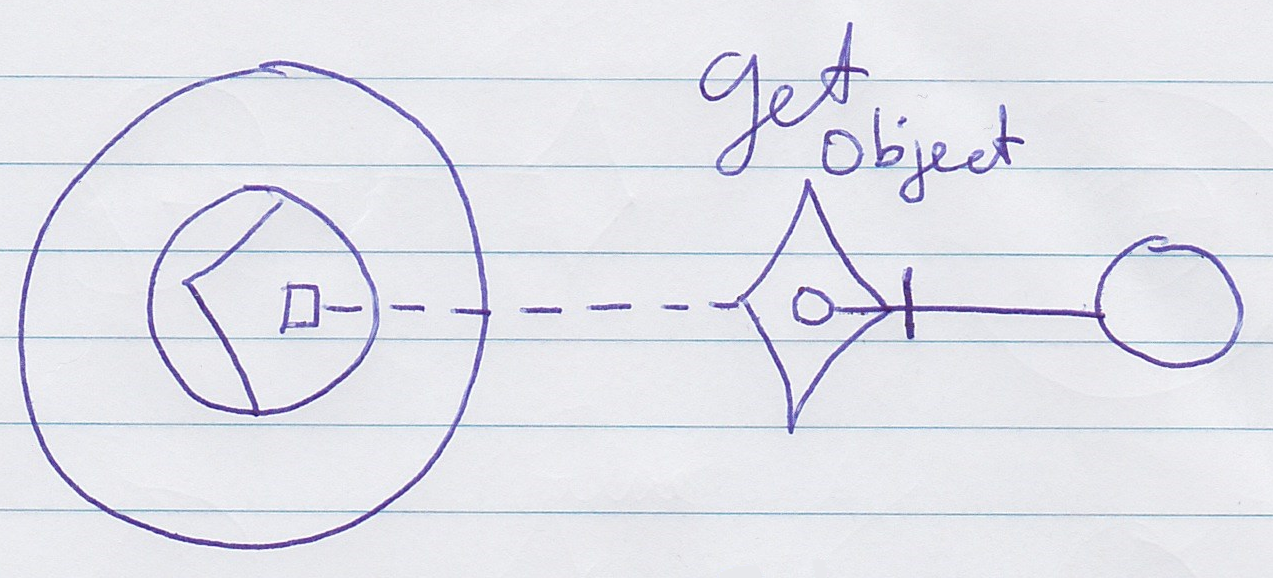
Here is an attempt to systematically display all the system commands thusfar in a system interface command call notation. Where the calls look similar this might be relevant to realize, where the calls look particular this might be relevant to realize as different. Which are similar and which are different might become apparent from this full overview.

|  |  |
| --- | --- |
|  |  |
| (has assignment notation) | (has assignment notation) |
|  |  |
|  |  |
| (has assignment notation) | (has assignment notation) |
|  |  |
|  |  |
| (object-bound)  (has cross-aspect assignment notation) | (reference-bound)  (has cross-aspect assignment notation) |
|  |  |
|  |  |
| (has assignment notation) | (has assignment notation) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| (also has a basic command call notation) |  |

### System Interface Calls with Arguments

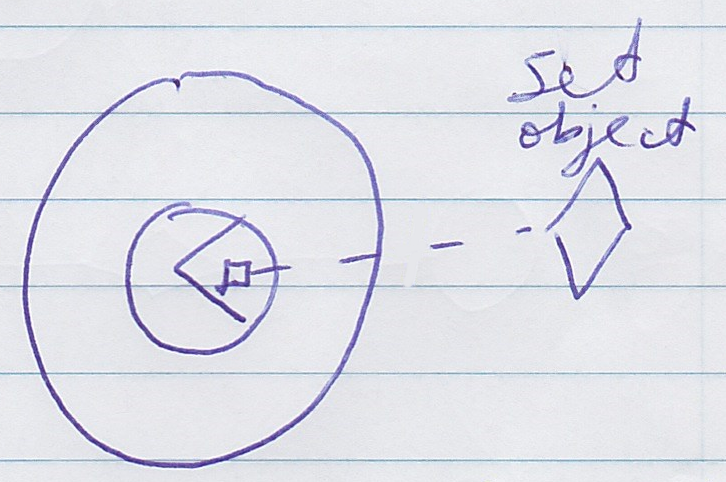
Above were shown the basic system command call notation, but most system command calls are incomplete without an argument.

The **Get Object** command has a return value: the object. It is an *output* argument. It should be there to transfer the **Object** aspect to another symbol. The **Get Object** command's return value is drawn out as follows.

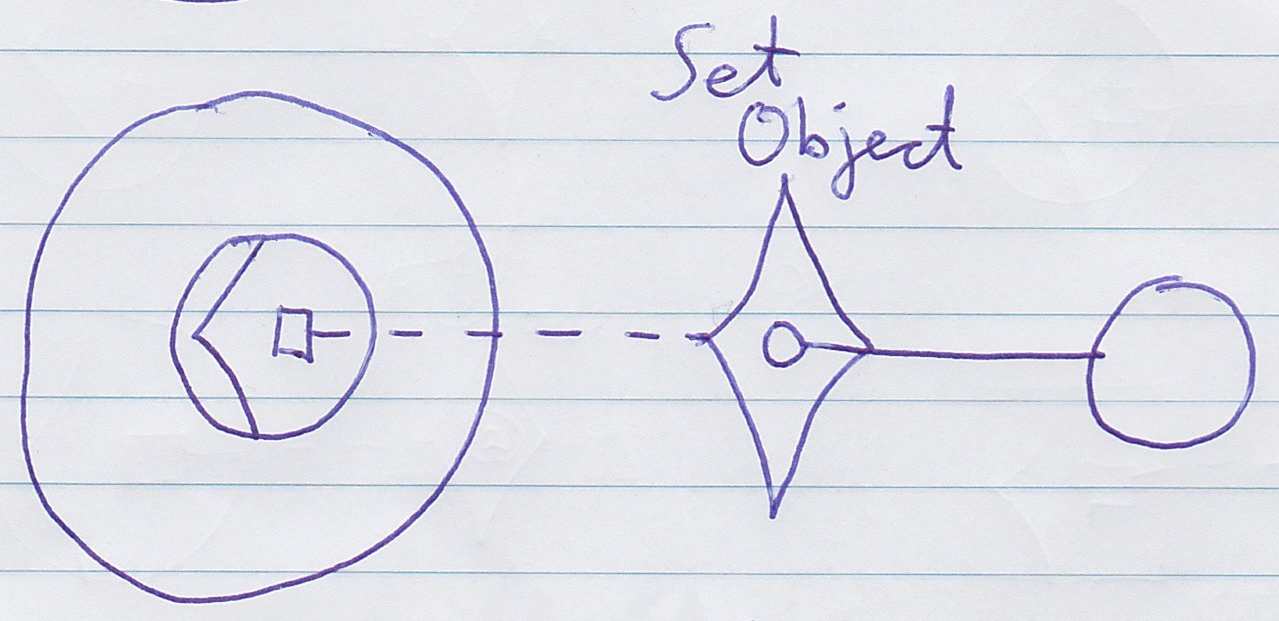


In the picture above the output argument of the **Get Object** call is filled in. **Get Object** is called upon the smaller circle contained by the larger circle. But next, an **Set Object** is *implicitly* called upon the smaller circle in the corner. The diagram above implicitly causes an **Set Object** to be executed upon the smaller circle in the corner.

Something similar happens in case of an **Set Object** command. Here is the basic notation of an **Set Object** command call:



And here the argument of the **Set Object** command is filled in:



This causes an implicit **Get Object** on the smaller circle in the top-right corner, followed by the explicit **Set Object**, targeting the smaller circle contained by the larger circle.

So the basic display of a basic system command call with an argument filled in, implictly causes another system command to be called as well.

Below you will find an overview of all the system commands with their argument filled in.

Which system commands might be called implicitly, is also indicated in the overview. The implicit calls might be regarded indications of what *might* happen or what system call the connection might have been established with. But perhaps not always. It might be a good way to go to have a relaxed view on what the connection means versus the implicit calls indicated.

Some commands might not be covered by assignment notation, which might make explicit calls needed. Others might be possible with an assignment notation also, which the table below also tries to denote.

|  |  |
| --- | --- |
| **Get Object 🡨**  (Set Object 🡪) | (Get Object 🡪)  **Set Object 🡨** |
|  |  |
| (has assignment notation) | (has assignment notation) |
|  |  |
| **Use As Class** (≈ Get Object) **🡨** (Set Class 🡪) | (Get Class 🡪)  **Set Class 🡨** |
|  | *(Might not applicable. There might not be a notation for an implicit Get Class call. System interface call with argument might require two system calls in this case: explicit Get Class and explicit Set Class.)* |
| (has assignment notation) |  |
|  |  |
| **Get Object-Bound Class 🡨**  (Set Class 🡪) | **Get Reference-Bound Class 🡨**  (Set Class 🡪) |
|  |  |
|  |  |
| (Use As Class (≈ Get Object) 🡪)  **Set Class 🡨** |  |
|  | - |
| (has assignment notation) |  |
|  |  |
| **Get Object-Bound Class 🡨**  (Set Object 🡪) | **Get Reference-Bound Class 🡨**  (Set Object 🡪) |
|  |  |
| (has cross-aspect assignment notation) | (has cross-aspect assignment notation) |
|  |  |
| **Value Get 🡨**  (Value Set 🡪) | (Value Get 🡪)  **Value Set 🡨** |
|  |  |
| (has assignment notation) | (has assignment notation) |
|  |  |
| **Get Clone (2) 🡨**  (Set Clone (2) 🡪) | (Get Clone (2) 🡪)  **Set Clone (2) 🡨** |
|  |  |
|  |  |
| (Use As Class 🡪)  **New 🡨** | (Get Reference-Bound Class 🡨)  **New 🡨** |
|  |  |
|  | *(variation where class for object not provided as argument, but defined by class of the reference)* |
| **Annul** |  |
| *(no parameters)* | - |
|  |  |
| (Get Object 🡪)  **Add 🡨** | **Remove** |
|  | *(no parameters)* |

### Explicit Get & Set

Assignment has a special, simplified notation covered in the *Assignment* article. But you can also display the **Get** and **Set** of the assignment explicitly. This notation is only required when there is no simplified notation available for what you want. However, explicit notation of **Get** and **Set** can always be used in *abstract* diagram expression just to make a point.

|  |  |
| --- | --- |
| **Get Object 🡨 Set Object 🡪** | **Use As Class** (≈ Get Object) **🡨 Set Class 🡪** |
|  |  |
|  |  |
| **Get Object-Bound Class 🡨**  **Set Class 🡪** | **Get Reference-Bound Class 🡨**  **Set Class 🡪** |
|  |  |
|  |  |
| **Get Object-Bound Class 🡨**  **Set Object 🡪** | **Get Reference-Bound Class 🡨**  **Set Object 🡪** |
|  |  |
|  |  |
| **Get Value 🡨**  **Set Value 🡪** | **Get Clone (2) 🡨**  **Set Clone (2) 🡪** |
|  |  |
|  |  |
| **Use As Class 🡪**  **New 🡨** | **Get Object 🡪**  **Add 🡨** |
|  |  |

The **Annul** and **Remove** commands' explicit notations were already covered by other sections.

|  |
| --- |
| **Get Reference-Bound Class 🡪**  **New 🡪** |
|  |
| The workings of the **New** command without a class argument might be internal implementation of an overload of the **New** command in the system interface. |

### Explicit Display of Assignment

An assignment has a simple notation:



But it is an implicit notation. Assignment needs such a simple notation, because it is one of the most basic kinds of commands in computer technology. Other sections have demonstrated how assignment can be displayed in a more explicit way. The current section goes as far as to display assignment completely explicitly.

The point of this article is to demonstrate how assignment works at system-level and also to demonstrate how impractical the implicit notation is.

The main thing explicitly displayed here is that the *system interface* is opened up, showing explicitly what aspects and system commands are called.

The pictures that follow might all represent the different ways to display an assignment, that differ by whether the **Get** or **Set** calls are displayed as explicit calls to a system interface, or as a connection or as an assignment call.

|  |
| --- |
| **Explicit Get Call with Argument** |
|  |
| **Get Object 🡨**  (Set Object 🡪) |
|  |
| **Explicit Set Call with Argument** |
|  |
| (Get Object 🡨)  **Set Object 🡪** |
|  |
| **Explicit Get & Set Calls** |
|  |
| **Get Object 🡨**  **Set Object 🡪** |

But the main point is: you have now seen how system operations work internally and you can also see that it is highly unpractical to express all the assignments explicitly like that.

You could also explicitly display the assignment call, which basically puts the whole thing inside a command:

|  |
| --- |
| **Assignment Call Implementation** |
|  |
| **Get Object 🡨**  **Set Object 🡪** |

But ofcourse a more practical notation is:

|  |
| --- |
| **Assignment Notation** |
|  |
| **Get Object 🡨**  **Set Object 🡪** |